# Упражнения: Да направим CRUD приложение без ORM

## Създаване на просто приложение

В рамките на това упражнение, ще създадем стъпка по стъпка малко CRUD конзолно приложнеие за управление на продукти в нашата база данни. Ще използваме SQL Server за БД.

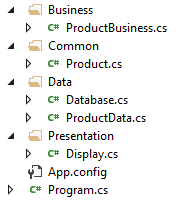
## Създаване на БД

Отворете SQL Server Management Studio и създайте таблица, която се нарича **shop**, а в нея създайте таблица:

|  |
| --- |
| **CREATE TABLE product**  **(**  **Id INT IDENTITY PRIMARY KEY,**  **Name VARCHAR(100) NOT NULL,**  **Price DECIMAL(10,2),**  **Stock INT**  **);** |

**Създаване нA структура на проекта**

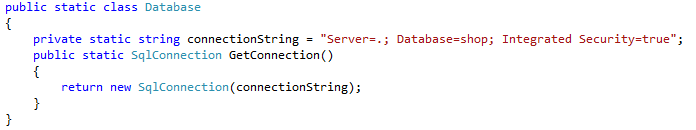
Нашият проект ще е организиран в трислойна архитектура, за тази цел създайте следните папки в проекта ви (може да използвате и проекти в solution-a):



## Слой за данни

В нашият стой за данни ще създадем два класа – **статичен** клас **Database.cs** и клас **ProductData.cs**

**Database.cs** ще поддържа връзката с БД – там ще се подава и **низа за връзка** към нея.



Освен това в този слой ще добавим и **ProductData.cs** в него ще създаваме необходимите SQL команди, за да извършваме **CRUD** операции.

Ще започнем реализирането с метода **GetAll()**, който трябва да върне списък от всички продукти в таблицата ни.

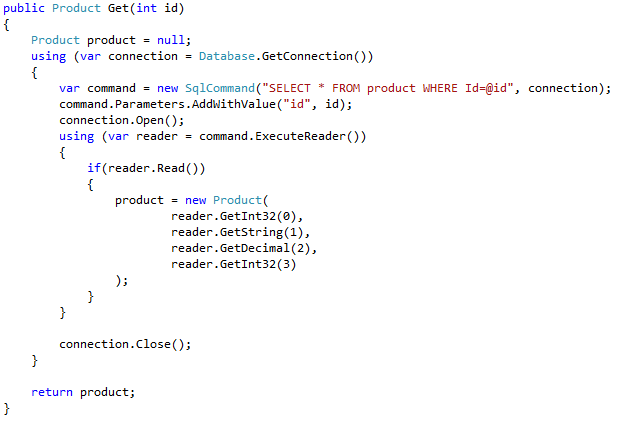
За целта ще ползваме SQL заявката за извличане на всичката информация от таблицата, а след това ще обработваме нейния резултат.

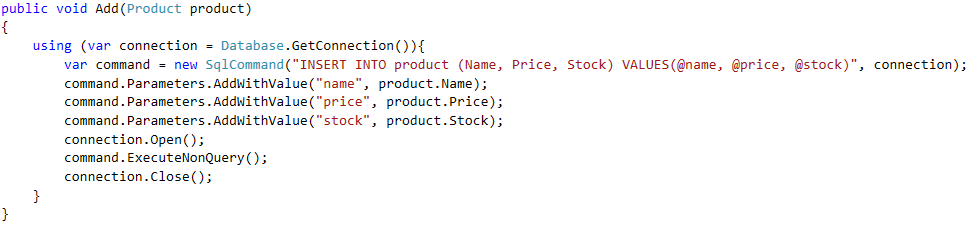
Това се случва чрез **ExecuteReader()** метод на **SqlCommand** класа и създаване на обект от клас **Product** на базата на информацията от четеца. Това се случва в **while** конструкция, която работи чрез метода **Read()** на четеца, който сме получили от **ExecuteReader().** Този метод на всяка итерация връща по един ред от таблицата. Когато накрая редовете в таблицата свършат, условието, което задвижва while конструкцията се превръща във **false** и така тя спира, а ние вече разполагаме с пълния списък на базата на тази таблица. Кодът е както следва:



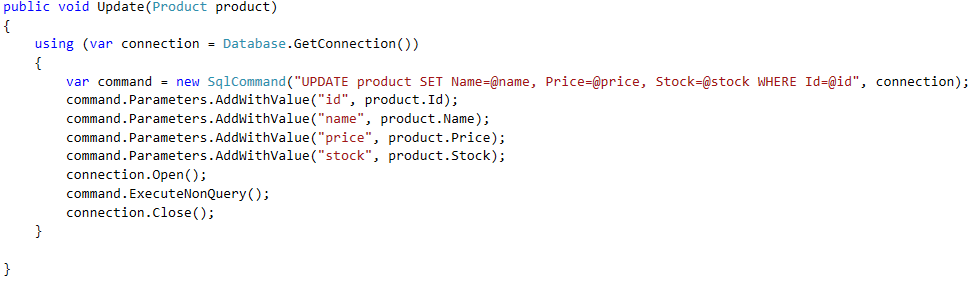
Следващият метод, с който ще продължим е **Get()** метода. В този метод искаме по подадено **id** на продукт да получим информацията за него. Тук ще използваме параметризирана заявка, където параметъра е нашият **id**.

След това чрез извикване отново на **ExecuteReader()** и **Read()** метода получаваме информация. Ако в нашата база данни няма продукт с желаното **id**, условието на **if-конструкцията** ще бъде **false** и по този начин **product** ще остане със стойност **null**, зададена му по начало. Кодът е както следва:

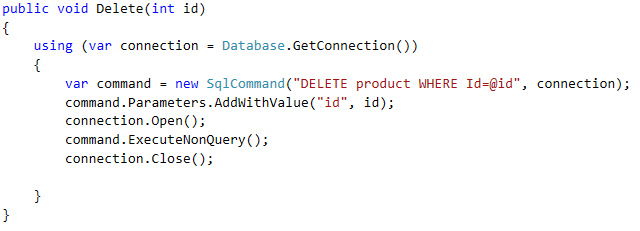


Сега нека да пристъпим към реализацията на **Add** метода. В него ще използваме параметризирана INSERT заявка. Реализацията на метода е доста праволинейна:  


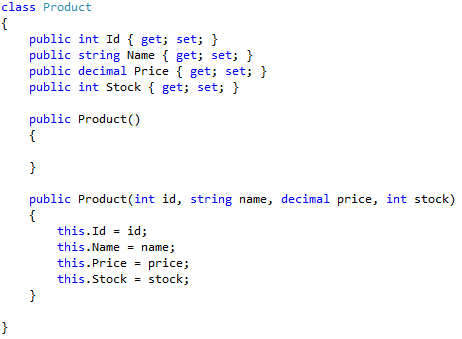
По сходен начин се реализира и Update метода. Разликата е, че в него ще използваме... UPDATE по Id ☺



Остана да реализираме метода за изтриване по id. Там ще използваме отново параметризирана заявка:

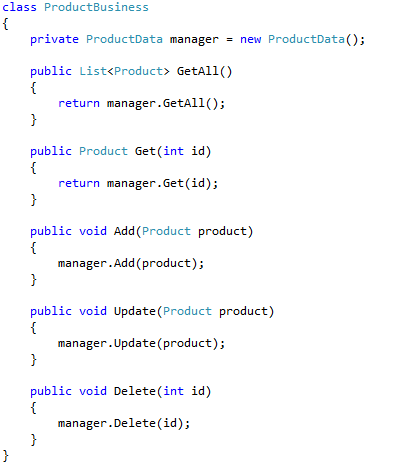


Навярно сте забелязали, че боравим с обекти от клас **Product**. Сега е време да го създадем. Тъй като този клас ще се явява общото между отделните елементи в нашата архитектура, той ще бъде в папката Common. В този клас няма нищо особено – просто свойства за 4-те колони от нашата таблица и конструктор. Вие вече сте реализирали такива класове, така че в този няма нищо ново за вас.



## Бизнес слой

Сега е време да реализираме класът с бизнес логиката на нашето приложение. Този клас сам по себе си е доста лесен за реализиране. Всичко, от което имаме нужда е поле от клас **ProductData** и методи, които викат тези на **ProductData** – **GetAll()**, **Get()**, **Add()**, **Update()**, **Delete()**:

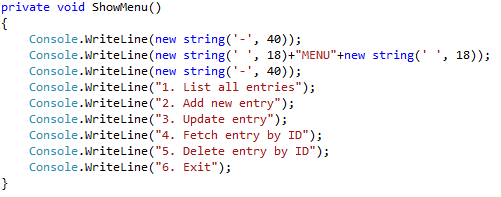


С това направихме проста реализация на нашата бизнес-логика, която предстои да използваме в Display.cs класа.

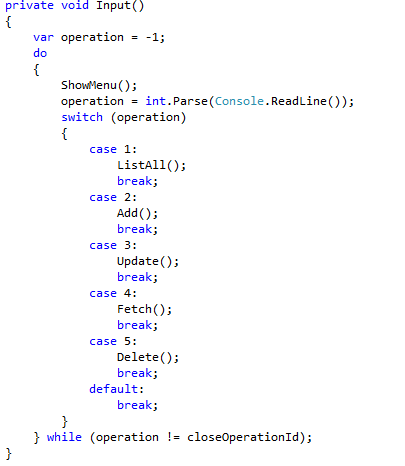
## Презентационен слой

Тук ще имаме Display.cs клас. Той ще реализира конзолно меню, от което ще въвеждаме желана опция и съответно по този начин ще бъде контролиран вход/изхода на програмата.

Ще започнем със създаването на частен ShowMenu() метод, който ще бъде викан, за да показва какви са възможностите:



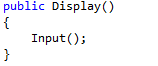
Този метод ще се вика от друг частен метод – **Input().** Целта на този метод е да получим вход от потребителя – номер на желаната операция и според това да извикаме някой от другите методи. Методът може да бъде реализиран, чрез позната за вас **do/while** конструкция и **switch/case** конструкция. Една примерна реализация на метода би изглеждала ето така:



В кода по-горе **closeOperationId** е поле на класа Display, в което задаваме номерът на операцията за затваряне на приложението, при въвеждането на който трябва да спрем да приемаме вход от потребителя:

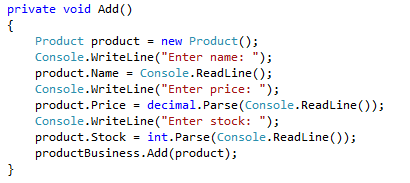


Самото извикване на **Input()** метода ще се случва в конструктора на класа Display:

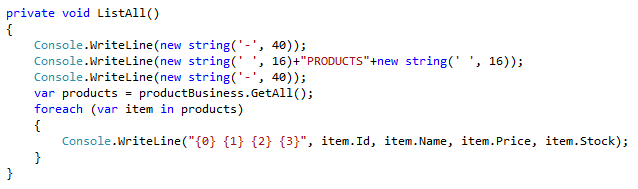


Сега трябва да реализираме останалите методи. Ще започнем с **Add().** Водещата задача през този метод е той да въвежда информация за нов продукт. Съответно в него ние в отделни променливи ще въвеждаме името, цената и наличността на продукта, след което ще създаваме обект от клас **Product** и чрез обектът **productBusiness** ще извикваме **Add()** метода на бизнес логиката.

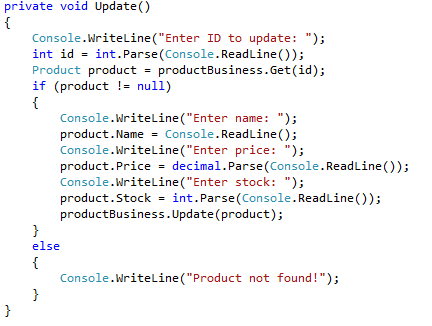
Кодът изглежда както следва:



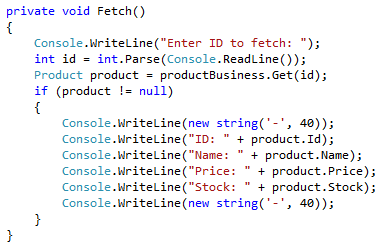
Следващият метод, който ще бъде реализиран е **ListAll().** В него трябва да си създадем променлива **products,** която ще получи своята стойност благодарение на метода **GetAll()** от бизнес логиката. След това трябва просто да обходим всички получени елементи и да ги изведем. Тук може да проявите въображение по начина на извеждане на елементите и да изведете информацията и по-красиво, тъй като примерната реализация не е направена особено красиво ☺



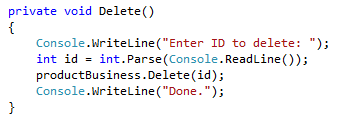
Сега, след като можем да виждаме вече всички елементи в таблицата, нека да преминем към възможността да ги редактираме. Това ще се случва в методът **Update(),** той ще иска от потребителя **id** на продукта за редактиране. Ако такъв продукт действително съществува, потребителят ще въвежда новите данни за него. Забележете, че този метод може да се направи значително по-удобен, отколкото в примера – например да показва досегашната стойност на свойствата в обекта... и не само ☺



След като направихме методът за редактиране, е време да направим и метод за визуализиране на информацията по **id** на даден продукт. Тук ще използваме **Get()** метода от бизнес логиката.



Накрая ще реализираме и метод за изтриване на продукт по неговото **id.** Тук отново ще използваме съответния метод от бизнес логиката:



За да завършим нашето приложение имаме нужда единствено от създаването на обект от клас **Display** в рамките на **Main** метода в **Program.cs.**

Така приключва нашата реализация на приложение без ORM. Плюсът тук е, че имаме по-голям контрол върху заявките и кодът прави **точно това, което сме му казали**. За сметка на това, ако желаем да добавим втора таблица **clients,** които да купуват продуктите, а след това и **orders,** в която да се записва поръчката на всеки клеинт, ще трябва да повторим част от тривиалните заявки, които написахме тук, а освен това ще трябва да пишем и доста допълнително код.

## Министерство на образованието и науката (МОН)

* Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".



* Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен** **лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).

